Haskell - Ein Einstieg

Wir haben uns bereits etwas mit Haskell auseinandergesetzt. Jetzt sollst du selbst mit Haskell programmieren. Rufe dazu die Seite https://repl.it auf (oder nutze den QR-Code rechts).
Öffne eine Haskell-Umgebung. Rechts findest du die Konsole, in der Mitte kannst du Skriptfiles erstellen.

① Nutze zuerst die Konsole um einfache Rechenaufgaben auszuprobieren.

- Beispiel:
  3 + 4
  3.0 + 4

② Nun ist das Standard-Anfangsprogramm an der Reihe. Erstelle ein Programmfile, das die Ausgabe „Hallo Welt.“ erzeugt.

- Nutze dazu die Funktion putStrLn().

③ Aus der letzten Woche kennst du die Funktion doubleMe, die einen Wert als Eingabe nimmt und das Doppelte ausgibt.
Um Fehler vorzubeugen, füge eine Mainroutine hinzu, wie rechts abgebildet.

```
main = do
putStrLn(„running“)
```

a) Schreibe die Funktion doubleMe in einen File.
b) Erweitere deinen File um eine Funktion doubleUs, die zwei Eingaben bekommt und das Doppelte der Eingabewerte summiert und das Ergebnis ausgibt. Nutze hierfür doubleMe.
c) Teste beide Funktionen.

④ Auch die Funktion doubleSmallNumber, die eine Zahl unter 100 verdoppelt, Zahlen über 100 und die 100 selbst aber unverändert ausgibt, ist bereits bekannt.

- Schreibe die Funktion doubleSmallNumber.
  Verwende hierfür die if-Struktur.

if-Abfrage:
if *Bedingung*
then *Aktion 1*
else *Aktion 2*

⑤ Definiere die Liste *numbers* vor der Main-Routine des Programms.
Teste anschließend die dir bereits bekannten Funktionen *tail*, *init*, *head* und *last*.

```
numbers = [1,2,3,4]
```

⑥ Definiere weitere Listen list1 bis list3 wie rechts angegeben oberhalb der Main-Routine.

a) Teste die Funktion *length*, indem du sie auf alle Listen anwendest.
b) Teste nun die Funktion *null*.
c) Teste die Funktion *reverse*.
d) Teste die Funktionen *take* und *drop*, indem du die Eingaben „take 3 list1“ und „drop 3 list1“ in der Konsole ausführst.
e) Teste die Funktionen *minimum*, *maximum*, *sum* und *product*.

```
list1 = [1..10]
list2 = []
list3 = [6,8,9,10,3]
```

Listen und Bedingungen

Um Listen erzeugen zu lassen, helfen Bedingungen.
Die Syntax ist [Rechenanweisung | Bedingungen für die Variablen].
Zum Beispiel erzeugt [x^2|x <- [1..10], mod x 2 ==0] eine Liste, in der die Quadrate der geraden Zahlen zwischen 1 und 10 enthalten sind.

⑦ Erzeuge Listen, die die folgenden Bedingungen erfüllen:

a) es sind nur gerade Zahlen enthalten, die kleiner sind als 10
b) es sind nur ungerade Zahlen enthalten, die zwischen 10 und 20 liegen
c) es sind alle Zahlen enthalten, die zum Einmaleins der 3 gehören (also bis 30).
d) es sind alle Zahlen zwischen 0 und 100 enthalten, die sowohl durch 7 als auch durch 5 teilbar sind.

⑧ Die Verflixte 7 ist ein Spiel für die Grundschule, bei dem die Kinder das Einmaleins der 7 üben, indem sie beim Zählen alle Zahlen durch *Piep* ersetzen, die durch 7 geteilt werden können.
Erzeuge eine Liste der Zahlen von 1 bis 70, wie sie im Spiel Verflixte 7 entsteht, wobei der Einfachheit halber die durch 7 teilbaren Zahlen durch 0 ersetzt werden sollen.

⑨ Erzeuge eine Liste, die alle Primzahlen enthält, die kleiner als 50 sind.

- Beginne mit einer Liste der Zahlen bis 50.
- Filtere alle Zahlen aus, die keine Primzahlen sind, weil sie durch eine Primzahl geteilt werden können.
- Füge bei der Filterung Ausnahmen hinzu, so dass die Primzahlen selbst **nicht** ausgefiltert werden. Verwende dafür die logischen Operatoren und && und oder ||.
- Es genügt die Primzahlen bis 11 in der Filterung zu verwenden.

Primzahlen:

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47

Tupel

Aus der Mathematik kennst du bereits Tupel. Dir sind Paare und Tripel bekannt, zum Beispiel als Koordinaten von Punkten in der Ebene oder im Raum. Haskell kann ebenfalls Tupel bilden. Die Anzahl der Elemente ist dabei nicht eingeschränkt und es können auch unterschiedliche Typen verwendet werden, zum Beispiel (1, Amsel).
Die Schreibweise unterscheidet sich nicht von der Mathematik. Tupel werden durch runde Klammern umschlossen, die Reihenfolge der Elemente kann nicht beliebig vertauscht werden. Die einzelnen Elemente werden durch Kommata getrennt.

⑩ Definiere die Tupel p1 (1, 2, 3) und p2 (2, 4, 6) oberhalb der Main-Routine.

a) Schreibe die Funktion add (x1, y1, z1) (x2,y2,z3), die die Elemente der beiden Tupel addiert.
b) Schreibe weitere Funktionen sub und mult, die die Elemente subtrahiert bzw. multipliziert.
c) Weshalb ist eine solche Funktion für die Division nicht ohne weiteres möglich?

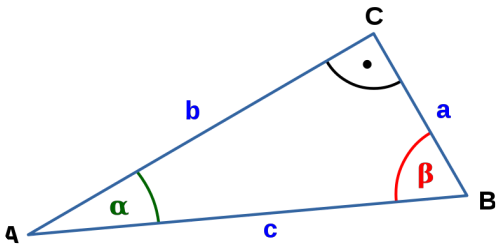
(11) Nutze Haskell um folgende Aufgabe zu lösen:
Erstelle eine Liste von Dreiecken, deren Seitenlängen zwischen 1 cm und 10 liegen.

- Verwende ein Tupel (a,b,c), das als Dreieck mit den Seitenlängen a, b und c verstanden wird.
- Weise jeder Variablen in der Bedingung der Liste ihre möglichen Werte zu.

(12) Die Aufgabe 11 enthält einen Logikfehler. Worin besteht er?

(13) Nutze Haskell um folgende Aufgabe zu lösen:
Erstelle eine Liste von rechtwinkligen Dreiecken, deren Seitenlänge zwischen 1 cm und 10 cm liegen.

- Dreiecke sind rechtwinklig, wenn in ihnen die Seitenbeziehung $a^2 + b^2 = c^2$ gilt.



(14) Nutze Haskell um folgende Aufgabe zu lösen:
Ein rechtwinkliges Dreieck hat einen Umfang von 24 cm. Die Seitenlängen sind ganzzahlig und bewegen sich zwischen 1 cm und 10 cm.
Welche Seitenlängen sind möglich?

- Den Umfang eines Dreiecks berechnet man mit U = a+b+c.
- Als zusätzliche Schwierigkeit sorge dafür, dass spiegelbildliche Dreiecke nur einmal ausgegeben werden.

(15) Schreibe eine Funktion *dist*, die als Eingabe zwei Punkte bekommt und den Abstand dieser beiden Punkte zueinander bestimmt.

- Beschränke dich auf die Ebene, Punkte haben also eine x- und eine y-Koordinate.
- Den Abstand eines Punktes P_1 von einem Punkt P_2 berechnet man mit der folgenden Formel:
  $d^2 = (x_1-x_2)^2 + (y_1-y_2)^2$, wobei d der Abstand ist.
  Es handelt sich hierbei um eine Anwendung des Satzes des Pythagoras.
- Die Wurzel berechnet man in Haskell mit der Funktion *sqrt*.
- Da die Wurzel eine Kommazahl ausgibt, möchte sie einen entsprechenden Datentyp als Eingabe bekommen. Nutze dafür die Funktion *fromIntegal*.
- Anders als wir es gewohnt sind, schachtelt Haskell Funktionen, indem Klammern gesetzt werden, die **vor** dem Funktionsnamen beginnen und nach dem Argument enden.
  Zum Beispiel: sqrt (fromIntegral (x+y))